Applying differential dynamic logic to reconfigurable biological networks

Daniel Figueiredo, Manuel A Martins, Madalena Chaves

This is a preliminary author version of the article published as:
D. Figueiredo, M.A. Martins, and M. Chaves. Applying differential dynamic logic to reconfigurable networks, Mathematical Biosciences, 291, pp. 10-20, 2017.

Applying *differential dynamic logic* to reconfigurable biological networks

Daniel Figueiredo, Manuel A. Martins

CIDMA - *Center for R&D Mathematics and Applications*
Department of Mathematics, University of Aveiro, Portugal

Madalena Chaves

INRIA Sophia Antipolis - Mediterranée
and Université Côte d'Azur, France

Abstract

Qualitative and quantitative modeling frameworks are widely used for analysis of biological regulatory networks, the former giving a preliminary overview of the system's global dynamics and the latter providing more detailed solutions. Another approach is to model biological regulatory networks as hybrid systems, i.e., systems which can display both continuous and discrete dynamic behaviors. Actually, the development of synthetic biology has shown that this is a suitable way to think about biological systems, which can often be constructed as networks with discrete controllers, and present hybrid behaviors. In this paper we discuss this approach as a special case of the reconfigurability paradigm, well studied in Computer Science (CS).

In CS there are well developed computational tools to reason about hybrid systems. We argue that it is worth applying such tools in a biological context. One interesting tool is Differential Dynamic Logic (d$\mathcal{L}$), which has recently been developed by Platzer and applied to many case-studies. In this paper we discuss some simple examples of biological regulatory networks to illustrate how d$\mathcal{L}$ can be used as an alternative, or also as a complement to methods already used.

Keywords:
Differential dynamic logic, Biological regulatory networks, Hybrid systems, Discrete controllers

1. Introduction

Biological systems have been subjected to a intense study due to their importance to our lives and environment. Scientists study biological regulatory networks in order to better understand the dynamics of a cell, validate experimental results, find patterns or predict behaviors. Modeling and simulation techniques are widely used nowadays but there is still a limited number of variables and details that is possible to compute. Moreover, qualitative models like Boolean networks, which are easier to deal with, often are too abstract and, quantitative models must be applied.

Despite the degree of precision gained by using quantitative models such as systems of differential equations or stochastic equations, there are two main disadvantages in using these kind of models: first, the difficulty in performing analytical studies and, second, the necessity to rely on simulation and computational tools, which are limited by the continuous nature of the variables and the consequent heaviness of the numerical algorithms used to solve the equations. Furthermore, very often biological systems admit reconfigurable behaviors, that suggests considering both continuous and discrete dynamics of a system. For instance, a population of bacteria *E. coli* is observed to "switch" between two growth rates as the form of available sugar changes from e.g., glucose to lactose [1]. Indeed, for each

type of sugar, the genetic network of the bacteria may adopt different configurations: it is well known that the *lac* operon genes are expressed only if the sugar is lactose; otherwise their transcription is turned off [2].

The term reconfigurable has been used to classify computational systems whose functionality may change from one mode to another in response to external stimulus from the environment or from interactions with other systems. Such systems can alternate between several configurations (modes of operating). At present, most software systems have components that frequently reconfigure. For example, the cloud based applications modify its functionality to client demands and car cruise control can either be turned on or turned off changing the functionalities available. Mathematical foundations to deal with reconfiguration of systems have been intensively studied. In [3] a theory based on institutions is proposed, in [4] the methodology is applied to automatic insulin pumpers, and in [5] bisimulations are used to study some main features of Boolean network models. In this paper we explore the concept of reconfigurability and its possible realizations in biological systems, by presenting several illustrative examples. Some ways to identify and/or define reconfigurability in biological systems have already been discussed by several researchers in the context of molecular logic. For example, in [6] Prasanna De Silva arguments that molecules can be "interrogated" in many ways. In [7], the authors define reconfigurability of a system as the capability of *multifunctionality*, *i.e.*, the capability of some devices to switch between different operating modes. This capability occurs due to several factors, for instance, by decreasing a threshold above which the required output is present. Reconfigurable behavior may also be related to physical or internal changes in the system: for example, "re-wiring" or reorganizing the connections between its modules induces the system to respond differently to the same given input (see [8]). Many other examples appear in [6].

Many physical or mechanical systems are regulated by discrete controllers [9], which introduce "jumps" or switches at appropriate instants in the continuous behavior. These switches can be viewed as reconfigurations of the continuous system. Systems whose dynamics incorporates both discrete and continuous evolutions are called *hybrid*. These kinds of systems can be widely found in our life, for example cruise control in cars and the autopilot in planes are both hybrid systems, obtained by applying discrete controllers to purely continuous systems. In biological context, discrete controllers can be seen, for instance, as an insulin pump (that induces a variation in the concentration of insulin in a discrete way) and can be useful to avoid a system to reach an undesirable configuration. In general, hybrid systems may interchange between two different configurations (hence between different dynamical behavior) in response to a discrete time controller. Since neither purely discrete nor purely continuous models can completely represent the dynamics of hybrid systems, tools and logics able to deal with the hybrid features are needed [9, 10]. In this direction, we propose the application of Differential Dynamic Logic (d$\mathcal{L}$) to reason over biological systems (d$\mathcal{L}$ is a dynamic logic that embeds a first-order structure and whose syntax is interpreted over the reals. This logic was introduced by A. Platzer in [11] and intensively studied and applied to important case-studies by him and his collaborators).

When studying a hybrid system, its dynamics can be described by a hybrid program in $\mathcal{L}$, and its properties by a formula of this logic. Moreover, we can use the sound proof calculus of d$\mathcal{L}$ to find formal proofs/counterexamples for the properties described. Indeed, some applications of this logic to study hybrid systems can be found in [12], [13] and [14]. Additionally, there is a computational tool called *KeYmaera*. This tool is a semi-automatic prover over the syntax of d$\mathcal{L}$, i.e., given a property that is expected to hold by a formula of d$\mathcal{L}$, *KeYmaera* tries to prove that the property indeed holds. Sometimes, *KeYmaera* is not able to prove a (true) formula, in which case it will ask for help from the user at some point. However, if the property described by the formula of d$\mathcal{L}$ is not valid, *KeYmaera* can often lead us to counterexamples, since it presents the points it was not able to prove.

Many useful formal methods, and corresponding software, are available for studying biological regulatory networks, based on temporal logic, model checking, Petri nets, cellular automata, pi-calculus and other theoretical results (see the book [10] for a large sampling and review). Relative to these methodologies, d$\mathcal{L}$ logical approach has the advantage of representing continuous models without specifying values for the free parameters and verifying properties without abstracting any important characteristics from the model. The approach used in temporal logic [15] requires, in some sense, a discretization of the system, while the d$\mathcal{L}$ approach can deal with the system purely continuous or as a hybrid system. *KeYmaera* and its implementation of d$\mathcal{L}$ have also other advantages as: possibility to include time con-

straints or explicitly describe time evolutions; taking into account possible delayed controller reactions to bridge the gap of continuous-time models and discrete-time control design; and allowing nonlinear behavior and nontrivial reset relations, beyond the capabilities of linear hybrid automata. Thus, in this work we aim to corroborate the belief that d$\mathcal{L}$ is suitable also for biological systems and that it can be a good alternative (or a complement) to other models and tools already used in this area.

Outline. We start with a short introduction to the syntax and semantics of d$\mathcal{L}$. In Section 3 we discuss the notion of reconfigurability within hybrid systems and present some examples indicating that reconfigurable behaviors occur often in biochemical systems. Section 4 presents several examples to illustrate the application of d$\mathcal{L}$ logic to describe and prove properties of biological models.

2. Background: Differential dynamic logic

Throughout this paper, we assume that the reader is familiar with the concepts of modal logic, dynamic logic and first-order logic (cf. [16] and [17]). For easier reference, a few useful properties and concepts are recalled next.

2.1. Modal Logic

Modal logic is an extension of classical propositional logic which embeds the concept of "modalities". These modalities are introduced by two logical operators – $\Box$ and $\Diamond$ – that can be interpreted as "it is necessarily true that" and "it is possibly true that", respectively. Moreover, we have the semantical equivalence: "$\Box \equiv \neg\Diamond\neg$".

The models in modal logic are known as *Kripke models* and consist in triples (W, R, V), where W is a non-empty set of "worlds" (states), $R \subseteq W \times W$ is the accessibility relation between worlds and $V : \Omega \to 2^W$ is a valuation function which maps every atomic proposition $p \in \Omega$ (the set of atomic propositions) into the set of all worlds where the atomic proposition p is valid. This models can be seen as digraphs with a valuation function. Thus, this kind of models can be very useful to model state transition machines.

About the semantics of modal logic, we write $M, w \vDash \varphi$ meaning that φ is valid at a world (or state) w of a model M. The semantics of Boolean operators are as expected and, therefore, the novelty comes from operators $\Box$ and $\Diamond$. The meaning of the formula $\Box\varphi$ is "all successor states verify φ" and $\Diamond\varphi$ is interpreted as "there is one successor state which verifies φ. Formally, we have:
- $M, w \vDash \Box\varphi$ if $M, v \vDash \varphi$ for all v such that $(w, v) \in R$ and
- $M, w \vDash \Diamond\varphi$ if $M, v \vDash \varphi$ for some v such that $(w, v) \in R$.

In practical cases we sometimes need more than one modality to represent different transitions or events. The definitions can be extended straightforward for that case and the resultant logic is called *multimodal*.

Example 2.1. In the example on Figure 1 it is represented a model $\mathcal{M} = (W, R, V)$ with four worlds/states. The arrows represent the transitions described by the relation R and the letters p and q within each world represent the valid atomic propositions on each state. Therefore we have $W = \{w_1, w_2, w_3, w_4\}$, $R = \{(w_1, w_2), (w_1, w_3), (w_2, w_4)\}$ and $V : \{p, q\} \to 2^W$ such that $V(p) = \{w_1, w_2, w_3\}$ and $V(q) = \{w_1, w_4\}$. We can observe that $\mathcal{M}, w_1 \vDash \Box p$, because both $\mathcal{M}, w_2 \vDash p$ and $\mathcal{M}, w_3 \vDash p$. We also have $\mathcal{M}, w_3 \nvDash \Diamond q$, because there is not any other world $w \in W$ such that $(w_3, w) \in R$ and $\mathcal{M}, w \vDash q$. Moreover, although $\mathcal{M}, w_3 \nvDash \Diamond q$, we can verify that $\mathcal{M}, w_1 \vDash \Diamond\Diamond q$ since $\mathcal{M}, w_2 \vDash \Diamond q$. Finally, we point out that the formula $(p \wedge \neg p)$ is false in every world but $\mathcal{M}, w_4 \vDash \Box(p \wedge \neg p)$. This holds because, by definition, $\Box(p \wedge \neg p)$ holds at w_4 if $p \wedge \neg p$ is valid on each successor state of w_4. Since there is no sucessor state of w_4, $\mathcal{M}, w_4 \vDash \Box(p \wedge \neg p)$.

Next example shows that modal logic can be used to express properties of biological Boolean models.

Example 2.2 (Bistable switch example)**.** Let us consider a bistable switch synchronous Boolean model given by the following update functions:

$$\begin{cases} x^+ = \neg y \\ y^+ = \neg x \end{cases}$$

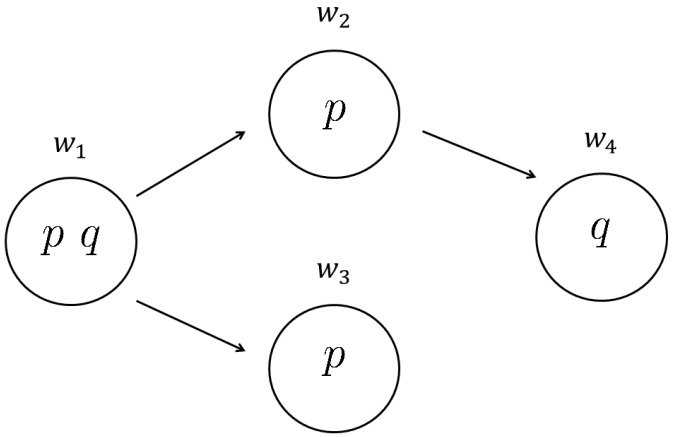

Figure 1: Kripke model $\mathcal{M}$.

This classical example represents a system with two components such that the presence of each one inhibits the production of the other. If the concentration of the first component (that we denote by x) is too high, then the second one (whose concentration we denote by y) will not be produced and its concentration will decrease due to degradation. Analogously, the same happens to x if the concentration of y is high enough. Therefore, the Boolean model presented in Figure 2 represents the four possible configurations of the system. For instance, the state 00 represents the configuration where x and y are small enough in order not to inhibit the production of the other component. Each state of the system

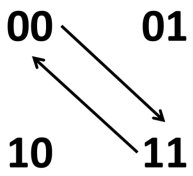

Figure 2: Boolean model of the bistable switch example.

is determined by the value of the Boolean variables x and y. Therefore, we can express the transitions between states by the formulas $(x \wedge y \rightarrow \Diamond(\neg x \wedge \neg y))$ and $(\neg x \wedge \neg y \rightarrow \Diamond(x \wedge y))$. The first formula means: "if at some state both x and y are 1 (*true*), then there is a transition to a state where both x and y are 0 (*false*). Similarly, the second says that "if at some state both x and y are 0 (*false*), then there is a transition to a state where both x and y are 1 (*true*). In particular, we note that the synchronous biological Boolean model of the bistable switch example presented verifies these formulas.

2.2. Dynamic Logic

Dynamic logic is an extension of modal logic that includes programs which can be used to describe the dynamics of diverse kinds of systems. Each program is used to describe different modalities which correspond to the execution of the respective program.

To define the set of admissible programs P we need to consider a set Π_0 of atomic programs. Thereafter, the set P is obtained by combining these atomic programs using specific operators. Specifically, if α and β are two programs of P, then $\alpha \cup \beta$, $\alpha;\beta$, α^* are in P. Moreover, for any formula φ of dynamic logic, $?\varphi$ is also a program in P. In practice, the execution of $\alpha \cup \beta$ consists in randomly executing either the program α or the program β and then the program terminates; the execution of $\alpha;\beta$ expresses a behavior in which β starts after α finishes, and then the program terminates; the execution of α^* consists in executing α a finite random number of times and it terminates afterwards. Finally, the execution of $?\varphi$ is used to verify if the formula φ is valid in the present state. If so, the program terminates successfully, otherwise, it does not.

The usual programming language constructors can be expressed within the syntax of dynamic logic:

if φ then π	$(?\varphi);\pi$
if φ then π_1, else π_2	$((?\varphi);\pi_1) \cup ((?\neg\varphi);\pi_2)$
while φ do π	$((?\varphi);\pi)^*;(?\neg\varphi)$
break	$(?\bot)$

Note that the program "break" never terminates successfully. We also point out that the program corresponding to the "while" terminates neither after nor before the first time φ becomes false because the tests would break the execution and, therefore, the program would not terminate successfully.

Dynamic logic has still limited expressiveness, however its syntax is the basis for several other logic systems and it is fundamental to express the dynamics of several of systems as those we present further in this paper (see [18]).

The syntax and semantics of dynamic logic are similar to multimodal logic. The main difference is that the programs (modalities) are structured by some constructors and, although each modality is labeled by a program – $[\pi]$, $\langle\pi\rangle$, the interpretation of the non atomic programs is determined by the interpretation of the atomic components of them, by means of the constructors. A formula like $[\pi]\varphi$ is interpreted as "whenever the program π is executed and terminates successfully, φ is true" and $\langle\pi\rangle$ must be interpreted as "it is possible that after π be executed and terminated successfully, φ is true". Because of this, the formula $[?\bot]\varphi$ is always true because, by definition, it is true if whenever the program $?\bot$ is executed and terminates successfully, φ is true. However, $?\bot$ (break) never terminates successfully and, thus, $[?\bot]\varphi$ is always true. Analogously, $\langle?\bot\rangle\varphi$ is always false.

2.3. Differential Dynamic Logic

Here we review the basic notions of *Differential Dynamic Logic* (d$\mathcal{L}$). We suggest to the reader reference [11] for a deeper understanding of d$\mathcal{L}$. This logic is a dynamic logic which considers two kinds of atomic programs:

- Discrete Jump Sets – $x_1 := \theta_1, \ldots, x_n := \theta_n$;
- Continuous Evolutions – $x_1' = \theta_1, \ldots, x_n' = \theta_n$.

Discrete Jump Sets (discrete assignment) and the Continuous Evolutions (defined by a system of ordinary differential equations) describe discrete and continuous evolutions, respectively. Combining both kinds of atomic programs it is possible to describe hybrid behaviors (see [11]).

2.3.1. Syntax

The syntax of d$\mathcal{L}$ considers a set V of *logical variables*, which can be quantified, and a set Σ (a *signature*). Σ is the signature of real numbers with symbols for state variables, that is, it contains:

- *Predicate symbols*, such as $>$, $<$, $=$.
- *Function symbols*, such as $+$, $-$, $\cdot$, / and constants (functions with arity 0) such as 0 and 1. Skolem functions are also considered.
- *State variables*, which are constants whose semantical interpretation can vary during the evolution of the system. We write Σ_{fl} to refer the set of state variables.

We point out that although state variables are defined as constants, it is important to bear in mind that they have particular features as presented below. We proceed now with several definitions in order to define the set of formulas of d$\mathcal{L}$.

We denote by $Trm(V,\Sigma)$ the set of terms of d$\mathcal{L}$ and it is defined recursively as the smallest set that contains V, Σ_{fl} and such that $f(t_1,\ldots,t_n) \in Trm(V,\Sigma)$ for any function symbol $f \in \Sigma$ with arity n (possibly 0) and any $t_1,\ldots,t_n \in Trm(V,\Sigma)$. The set of first-order formulas of d$\mathcal{L}$, $Fml_{FOL}(V,\Sigma)$, is defined recursively as the smallest set that contains $p(t_1,\ldots,t_n)$, $\bot$, $\top$, $\varphi\vee\psi$, $\varphi\wedge\psi$, $\varphi\to\psi$, $\varphi\leftrightarrow\psi$, $\neg\varphi$, $\exists x\varphi$, $\forall x\varphi$ for any predicate symbol p of arity n, any terms t_i, any $x \in V$ and any φ, $\psi \in Fml_{FOL}(V,\Sigma)$.

Considering hybrid programs, $HP(V,\Sigma)$ is the smallest set that contains $(a_1 := t_1,\ldots,a_n := t_n)$, $(a_1' = t_1,\ldots,a_n' = t_n \,\&\, \chi)$ and $?\chi$, for any $t_1,\ldots,t_n \in Trm(V,\Sigma)$, any $\chi \in Fml_{FOL}(V,\Sigma)$ and $a_1,\ldots,a_n \in \Sigma_{fl}$; and such that it contains $\alpha;\beta$, $\alpha\cup\beta$, α^* for any α, $\beta \in HP(V,\Sigma)$.

Finally we define the set of formulas of d$\mathcal{L}$, $Fml(V,\Sigma)$, as the smallest set that contains $p(t_1,\ldots,t_n)$, $\varphi\vee\psi$, $\varphi\wedge\psi$, $\varphi\to\psi$, $\varphi\leftrightarrow\psi$, $\neg\varphi$, $\exists x\varphi$, $\forall x\varphi$, $[\alpha]\varphi$ and $\langle\alpha\rangle\varphi$ for any $\alpha \in HP(V,\Sigma)$, any proposition symbol $p \in \Sigma$, any $t_1,\ldots,t_n \in Trm(\Sigma,V)$ and any $\varphi,\psi \in Fml(V,\Sigma)$.

2.3.2. Semantics

Before go further, we enhance that in the syntax three main classes of symbols are considered: predicate and function symbols ($\Sigma\backslash\Sigma_{fl}$) whose meaning is fix and the natural one (0, >, +, etc...); state variables (Σ_{fl}) whose meaning is fix at each state but can be modified by running a hybrid program; and logical variables (V) whose meaning can not be modified by hybrid programs but can be quantified.

To evaluate a formula of d$\mathcal{L}$ we have to consider:

- An *interpretation* I, that is a function, whose domain is $\Sigma\backslash\Sigma_{fl}$, and that interprets the predicate and function symbols as the respective predicate or function of reals arithmetic.
- A *state* v, that is a map $v : \Sigma_{fl} \to \mathbb{R}$. We denote the set of all states by $\mathrm{Sta}(\Sigma)$.
- An *assignment* η, that is a map $\eta : V \to \mathbb{R}$.

Let $x, d \in \mathbb{R}$ and let $f : \mathbb{R} \to \mathbb{R}$ be a function. We define $f[x \mapsto d]$ as:

$$f[x \mapsto d](y) = \begin{cases} d, & \text{if } y = x \\ f(y), & \text{otherwise.} \end{cases}$$

Given an interpretation I, an assignment η and a state v, we now define the *valuation function* $val_{I,\eta}(v, \cdot) : Fml(V, \Sigma) \to \{true, false\}$ for all first-order formulas of d$\mathcal{L}$, recursively. We firstly define this function for first-order formulas because their valuation will be needed to interpret the accessibility relation induced by modalities. Thereafter, we define the valuation function for the entire set of formulas. In order to simplify the notation, we also denote the function used to valuate terms with the same symbol.

For an interpretation I, an assignment η and a state v, we define $val_{I,\eta}(v, \cdot)$ for terms in the following way:

- $val_{I,\eta}(v, x) = \eta(x)$, for $x \in V$ and $val_{I,\eta}(v, u) = v(u)$, for $u \in \Sigma_{fl}$.
- $val_{I,\eta}(v, f(t_1, \ldots, t_n)) = I(f)(val_{I,\eta}(v, t_1), \ldots, val_{I,\eta}(v, t_n))$, for any function symbol f and any terms t_i, $i = 1, \ldots, n$.

For formulas we have,

- $val_{I,\eta}(v, p(t_1, \ldots, t_n)) = true$ iff $I(p)(val_{I,\eta}(v, t_1), \ldots, val_{I,\eta}(v, t_n)) = true$, for any n-ary predicate symbol p and any terms t_i.
- For Boolean combination of formulas it is defined as expected, for example: $val_{I,\eta}(v, \neg\varphi) = true \Leftrightarrow val_{I,\eta}(v, \varphi) = false$.

For quantified formulas:

- $val_{I,\eta}(v, \exists x\varphi) = true$ iff $val_{I,\eta[x\mapsto d]}(v, \varphi) = true$ for some $d \in \mathbb{R}$.
- $val_{I,\eta}(v, \forall x\varphi) = true$ iff $val_{I,\eta[x\mapsto d]}(v, \varphi) = true$ for any $d \in \mathbb{R}$.

Before given the valuation of formulas with modalities we must define the semantical interpretation for hybrid programs. The transitions induced by modalities are defined using the transition relation $\rho_{I,\eta}(.) \subseteq \mathrm{Sta}(\Sigma)\times\mathrm{Sta}(\Sigma)$, define by:

- $(v, w) \in \rho_{I,\eta}(x_1 := \theta_1, \ldots, x_n := \theta_n)$ iff $w = v[x_1 \mapsto val_{I,\eta}(v, \theta_1)] \ldots [x_n \mapsto val_{I,\eta}(v, \theta_n)]$;
- $(v, w) \in \rho_{I,\eta}(x_1' = \theta_1, \ldots, x_n' = \theta_n \ \&\ \chi)$ iff $\exists\ r \in \mathbb{R}_0^+$ and there is a function f: $[0, r] \to \mathrm{Sta}(\Sigma)$, called *flow*, such that:
  - $f(0) = v$ and $f(r) = w$;
  - f respects the differential equations, *i.e.*, for each variable x_i, the valuation of x_i at the state $f(t)$, $val_{I,\eta}(f(t), x_i) = f(t)(x_i)$ is continuous and has a derivative of value $val_{I,\eta}(f(t), \theta_i)$ in the interval $]0, r[$;
  - The value of the variables $z \in \Sigma_{fl}\backslash\{x_1, \ldots, x_n\}$ remains the same during the continuous evolution, *i.e.*, $f(t)(z) = v(z)$ for any $t \in [0, r]$;
  - Every state $f(t)$ verifies χ, *i.e.*, $val_{I,\eta}(f(t), \chi) = true$ for any $t \in [0, r]$;
- $\rho_{I,\eta}(?\chi) = \{(v, v) \in \mathrm{Sta}(\Sigma)\times\mathrm{Sta}(\Sigma) : val_{I,\eta}(v, \chi) = true\}$;
- $\rho_{I,\eta}(\alpha \cup \beta) = \rho_{I,\eta}(\alpha) \cup \rho_{I,\eta}(\beta)$;
- $\rho_{I,\eta}(\alpha\ ;\ \beta) = \{(u, w) : \exists v \in \mathrm{Sta}(\Sigma)$ such that $(u, v) \in \rho_{I,\eta}(\alpha)$ and $(v, w) \in \rho_{I,\eta}(\beta)\}$;

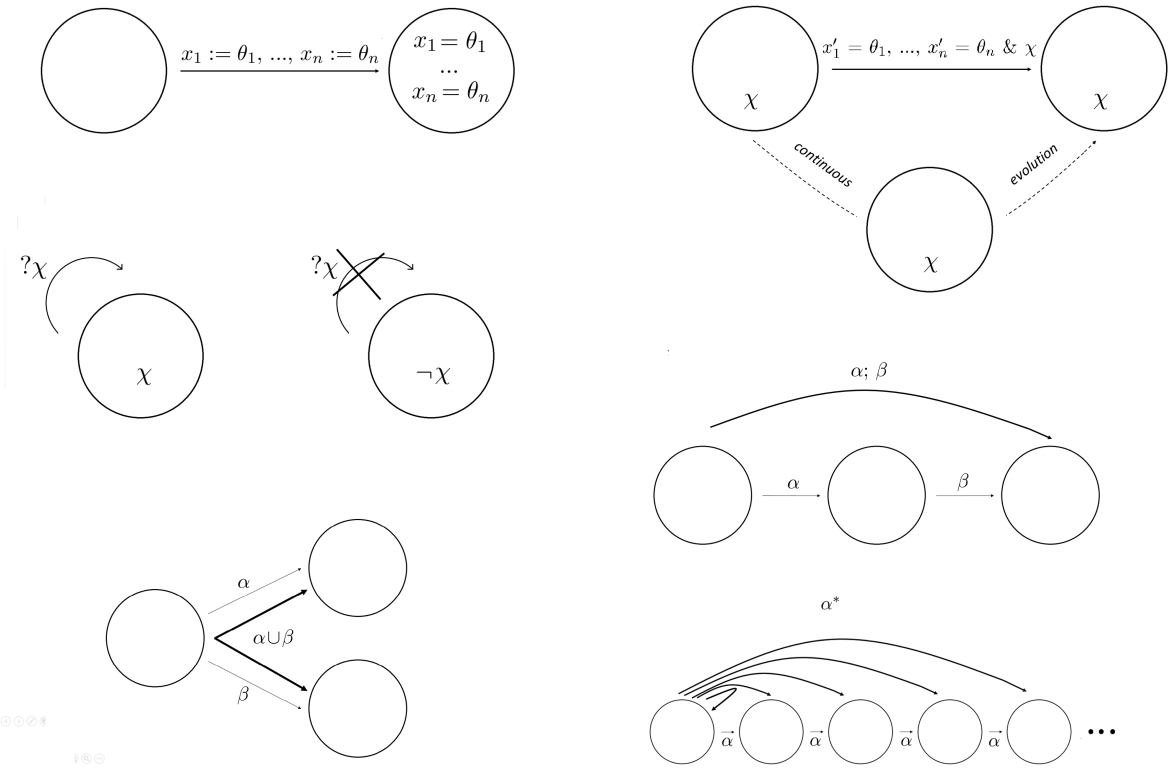

Figure 3: Illustration of the hybrid programs induced transitions.

- $\rho_{I,\eta}(\alpha^*) = \{(u, w) : \exists n \geq 0$ integer, $\exists v_0, \ldots, v_n \in \mathrm{Sta}(\Sigma)$ such that $u = v_0, w = v_n$ and $(v_{k-1}, v_k) \in \rho_{I,\eta}(\alpha)$ for any $k \in \{1, \ldots, n\}\}$.

An illustrative representation of transitions induced by each kind of hybrid program constructors is shown in Figure 3.

Next we present some examples in order to exemplify how can the semantics of d$\mathcal{L}$ be used to describe the dynamics of hybrid systems. Along the next examples x_i denotes state variables for $i \in \{1, \ldots, n\}$. A state u is represented by $(u_1, \ldots, u_n)$, where $u(x_i) = u_i$ for $i \in \{1, \ldots, n\}$. We note that if $(u, v) \in \rho_{I,\eta}(\alpha)$ for some states u, v and program α, then it means that it is possible to reach the state v from u by successfully executing α.

Let $J = \{j_1, \ldots, j_k\} \subseteq \{1, \ldots, n\}$ and α be a program defined by $x_{j_1} := a_{j_1}, \ldots x_{j_n} := a_{j_n}$, with $a_{j_1}, \ldots, a_{j_1}$ real numbers. A pair (u, v) belongs to $\rho_{I,\eta}(\alpha)$ if $u_k = v_k$ for $k \notin J$ and $u_k = a_k$ otherwise. In particular, since the execution of a discrete jump set is deterministic, each state u admits exactly one state v such that $(u, v) \in \rho_{I,\eta}(\alpha)$. If now we take α as $(x'_{j_1} = a_{j_1}, \ldots x'_{j_n} = a_{j_n} \ \& \ \chi)$, an element (u, v) is contained in $\rho_{I,\eta}(\alpha)$ if there exists a continuous evolution solution of the system of ODEs, which drives u into v and all intermediate states verify χ. Moreover, if we want to describe an unconstrained continuous evolution, we must consider χ such that $\chi \Leftrightarrow \mathit{true}$ and, in this case, we omit the first order formula χ. The following examples illustrate these situations.

Example 2.3 (Discrete Jumps)**.** If we take $u = (1, 2, 0)$ and α defined by $x_2 = x_1 - 2$. Then, we have that the unique state v such that $(u, v) \in \rho_{I,\eta}(\alpha)$ is $v = (0, 2, 0)$.

Example 2.4 (Continuous Evolutions)**.** Let $u = (-2, 4)$ and α the continuous evolution program $(x_1' = 1, x_2' = 2x_1 \,\&\, x_2 \geq 1)$. We can easily see that the solution of the ODE considered induces a continuous evolution described by $x_1 = (t-2)$ and $x_2 = (t-2)^2$ for positive t. Thus, $(u, (-1.5, 2.25))$ and $(u, (-1, 1))$ are contained in $\rho_{I,\eta}(\alpha)$ but $(u, (1, 1))$ is not. Although all considered states verify $x_2 \geq 1$, we cannot reach the state $(1, 1)$ from u without crossing, for example, the point $(0, 0)$ which does not verify $x_2 \geq 1$.

Now we are in conditions to define the valuation for entire set of formulas of d$\mathcal{L}$:

- $val_{I,\eta}(v, \langle\alpha\rangle\varphi) = true$ iff $val_{I,\eta}(w, \varphi) = true$ for some state w such that $(v, w) \in \rho_{I,\eta}(\alpha)$.
- $val_{I,\eta}(v, [\alpha]\varphi) = true$ iff $val_{I,\eta}(w, \varphi) = true$ for any state w such that $(v, w) \in \rho_{I,\eta}(\alpha)$.
- The valuation of formulas of the form $p(t_1, \ldots t_n)$, $\bot$, $\top$, $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi \to \psi$, $\varphi \leftrightarrow \psi$, $\exists x\varphi$ and $\forall x\varphi$ is done as defined for first-order formulas.

If $val_{I,\eta}(v, \varphi) = true$, we say that a formula φ is *satisfied* in I, η, v. Moreover, φ is said to be *valid* if it is satisfied in any triple I, η, v.

Next example illustrates the expressiveness of d$\mathcal{L}$ by describing two properties of a theoretical model with the syntax of d$\mathcal{L}$.

Example 2.5. Consider a ODE model described by the following system of differential equations:

$$\begin{cases} x' = 1 - y \\ y' = x \end{cases}$$

The syntax of d$\mathcal{L}$ can be applied to study reachability problems. For instance, we can express that a specific state is reachable starting from another specific state. Actually, this can even be generalized to regions. Indeed, we can describe the property "we can reach a state where $x, y > 0$ whenever the initial state verifies $x, y < 0$" by the d$\mathcal{L}$ formula $(x < 0 \wedge y < 0) \to \langle x' = 1 - y, y' = x\rangle x > 0 \wedge y > 0$.
We can also study properties which relate the variables among themselves. In fact, we can express the property "we can reach a state where $x = y$ starting at $(x, y) = (0, 0.1)$ without crossing any state where $x < 0$ or $y < 0$" by the formula $\langle x := 0, y := 0.1; (x' = 1 - y, y' = x \,\&\, x \geq 0 \wedge y \geq 0)\rangle x = y$.

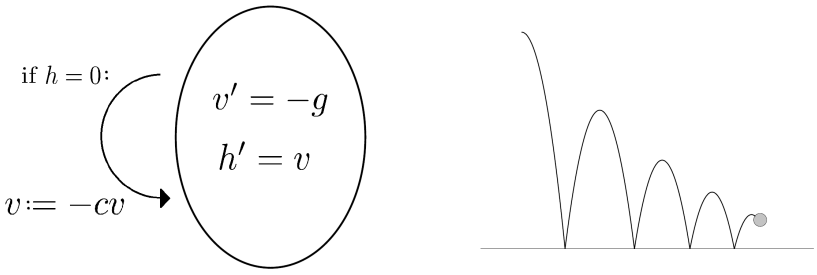

Figure 4: Bouncing ball example.

Example 2.6 (Bouncing Ball from [11])**.** In Figure 4 we can see that the dynamic of the ball in the air follows the laws of physics where the acceleration of the ball – derivative of the velocity v – is $-g$, the gravitational acceleration.The variable h is the height, which depends on the velocity of the ball, and t is the running time. In the instant the ball hits the ground, the velocity is instantaneously updated. At this point, the velocity becomes $-cv$ where the parameter c is an elasticity constant depending on the ball.

Neither ODEs nor discrete models are able to describe the dynamics of such system but the syntax of $d\mathcal{L}$ is able to do so. In order to do this, we must describe the behavior of the bouncing ball by an hybrid program of $d\mathcal{L}$.

The continuous evolution of the ball in the air is described by the ODE $v' = -g$ and $h' = v$.Thus, we must describe it by the program $(v' = -g, h' = v \,\&\, h \geq 0)$. We note that the continuous evolution is constrained by the condition $h \geq 0$ in order to assure that the ball does not go "below" the ground. Also, when the ball hits the ground we must describe its behavior by a discrete assignment $v := -cv$. Here, we can either consider that c has a fixed value or that it is a free logical variable. In order to be more general, in this example, we consider c a free logical variable. Then, we can describe the dynamics of bouncing ball by the following hybrid program:

$$((v' = -g, h' = v \,\&\, h \geq 0); ((?(h = 0); (v = -cv)) \cup (?(h \neq 0)))^*$$

This program admits a continuous evolution while $h \geq 0$ and, after that, the execution checks if $h = 0$. If so, it executes the discrete assignment to the velocity v. Anyway, this procedure is repeated finitely many times (in order to allow the ball to bounce arbitrary finitely many times). We note that, in this case we do not need the constrain into the ODE. Indeed, we can instead use the following hybrid program to describe the dynamics of this system:

$$((v' = -g, h' = v); ((?(h = 0); (v = -cv)) \cup (?(h > 0)))^*$$

Actually, this is possible because anytime $h < 0$ after the execution of the continuous evolution, the program will not be able to successfully terminate because it will fail either the test $?(h = 0)$ or $?(h > 0)$ and the execution will break.

More complex examples in the context of biological systems are presented in Section 4.

2.3.3. Proof calculus

For the reader interested in the logical details, we present some notions used to build a proof calculus. The proof calculus of d$\mathcal{L}$ uses sequents which are expressions with the form $\Phi \vdash \Psi$. In this representation we call the formulas in Φ the *antecedents* and the formulas in Ψ the *consequents*. We have the following useful equivalence:

$$\Phi \vdash \Psi \equiv \vdash \bigwedge_{\varphi_i \in \Phi} \varphi_i \rightarrow \bigvee_{\psi_i \in \Psi} \psi_i$$

Since the focus of this paper is not on logic, we do not present the rules of this proof calculus. They can be found in [11], as well as its basic properties. The proof calculus of d$\mathcal{L}$ is sound but incomplete, *i.e.*, although all provable formulas are valid, there are valid formulas which are not provable. This fact is a drawback, however, we can work around this problem. Indeed, in [11] a weaker result about completeness is proven. It states that the proof calculus of d$\mathcal{L}$ is complete if all valid formulas of the form $[\alpha]\varphi$, with $\alpha \in HP(V, \Sigma)$ being a Continuous Evolution and $\varphi \in Fml_{FOL}(V, \Sigma)$, are known.

In general, the process of constructing a proof is long and costly. Thus, it is important to have a tool to help us in this process. As referred before, for d$\mathcal{L}$ there is a software called *KeYmaera* which can produce the proof of valid formulas of d$\mathcal{L}$ in a semi-automatic way. More details about *KeYmaera* software can be found in [11].

3. Biological systems as reconfigurable systems

A cell contains many different components which are responsible for its regulation: mRNA, protein, enzymes, etc. These components interact with each other. For example, some protein A in the cell promotes the activation of a gene coding for another protein B, which in turn will inhibit the production of the first protein. Inside a cell, at each moment, several of these components are present at different concentrations and a great number of interactions are occurring. A biological regulatory network consists in a certain number of these components and the group of interactions between them.

Biological systems or modules frequently present reconfigurable behavior. As it happens in software design and in any complex software system, the modeling of biological systems require both expressive modeling languages and mathematically sound methods for development and verification of properties.

Regarding reconfigurations as transitions, we may propose some sort of modal logic as the language to express them. However the methods must be prepared to deal with whatever mathematical structures (and corresponding logics and languages) are used to formally describe a system's local configurations. Madeira in [3] presents the mathematical foundations of a (family of) logic(s) to reason about reconfigurable systems. The systems are modeled by generalized transition systems. In those structures, the transitions represent the changes between configurations and each world of the structure is a model of a specific configuration. It is important to refer that a logic (institution in his terminology) common to every state of the transition system must exist. This is precisely the approach we follow here. Hence, our approach can be described in a rather straightforward way: models for reconfigurable systems software are structured transition systems specified by an appropriate logical system. Their states are the individual configurations with whatever structure they have to be to model in concrete applications. On the other hand, transitions correspond to the admissible reconfigurations that can be performed in the real world.

There are several ways in which a system can be reconfigurable: discrete jumps imposed by physical constraints, external stimuli, or discontinuities in vector fields are some common occurrences. To identify reconfigurability in biological models, we thus propose a definition that uses the notion of *discrete event* [9], as a generalization of the "discrete jump sets" introduced in Section 2.3. A discrete event triggers a change in the current continuous system: it may represent a variable constraint ($h = 0$ in Example 2.6) or the addition of a (discrete) input with possible values $u \in \{0, 1, 2\}$. Each discrete event induces a transition from the "current configuration" to another "configuration": in general, there may be more then one transition allowed for each event.

Definition 3.1 (Reconfigurable System)**.** A reconfigurable system is a tuple $(X, \mathcal{V}, \mathcal{M}, \mathcal{Q})$ of nonempty sets such that:

- X is called the set of *state variables*;
- $\mathcal{V}$ is called the *evaluation set*;
- $\mathcal{M}$ is the set of *configurations* $M : D_M \times \mathbb{T} \to Sta(X)$, where
  - $\mathbb{T}$ is an additive monoid such as $\mathbb{R}_0^+$ (continuous case) or $\mathbb{Z}_0^+$ (discrete case).
  - $Sta(X)$ is the set of all *states* $v : X \to \mathcal{V}$. In case $X = \{x_1, \ldots, x_n\}$, we can identify a state v with the tuple $(v(x_1), \ldots, v(x_n))$;
  - $D_M \subseteq Sta(X)$ is the *invariant* of the configuration M;
  - $M(v, 0) = v$, for any state $v \in D_M$;
  - $M(M(v, t_0), t_1) = M(v; t_0 + t_1)$ for any state v and every $t_0, t_1 \in \mathbb{T}$;
- $\mathcal{Q}$ is a set of *discrete events* or *reconfiguration* (relations) $Q \subseteq (\mathcal{M} \times Sta(X)) \times (\mathcal{M} \times Sta(X))$, where
  - if $\big((M_1, v_1), (M_2, v_2)\big)$, $\big((M_3, v_3), (M_4, v_4)\big) \in Q$, then $M_1 = M_3$ and $M_2 = M_4$;
  - if $\big((M_1, v_1), (M_2, v_2)\big) \in Q$, then $v_1 \in D_{M_1}$ and $v_2 \in D_{M_2}$;
  - no pair with the form $((M, v), (M, v))$ is contained in Q.

We say that (v_0, v_1) is an *admissible M-evolution from v_0 to v_1* within the configuration M if $\exists\, \bar{t}, M(v_0, \bar{t}) = v_1$ and such that $M(v_0, t) \in D$ for all $t \in [0, \bar{t}]$.

Each configuration M is such that $M(v, t)$ represents the state reached from v after t units of time, according to respective configuration. The condition $M(M(v, t_0), t_1) = M(v; t_0 + t_1)$ guarantees the coherence of each configuration. In the examples we present in this paper, the configurations will be given using ODEs and discrete updating functions. Moreover, the first condition for the set of discrete events says that each discrete event Q can be seen as a relation between states of two configurations, formally (i) reconfigurations which do not are triggered at the same configurations are distinct and (2) reconfigurations which do not lead to the same configuration must be also distinct.

A discrete event Q, $(A, v), (B, v') \in q$ expresses the possibility of jump from the state v evolving according to the configuration A to the state v' evolving according to the configuration B by the discrete event q. We note that no pair with the form $((M, v), (M, v))$ is contained in any discrete event because, since these states changes occur instantaneously, that kind of pair does not represents any reconfiguration at all. The most tricky part will be to identify the appropriate set $\mathcal{Q}$ and the corresponding discrete events Q. The next examples illustrate this.

Anyway, one can consider a hybrid system. as a particular reconfigurable system. This is because, in this way, the continuous system admits an instantaneous (discrete time) reconfiguration. Bearing this in mind, we can also apply $d\mathcal{L}$ to the study of such systems (see Section 4). In [19], hybrid systems are formalized using the concept of hybrid automata. Therefore, we use that definition of hybrid automata and explain why can they be considered as a particular case of reconfigurable systems.

Definition 3.2 (Hybrid Automata)**.** A hybrid automaton is a tuple $(\mathcal{M}, E, \Sigma, X, init, inv, dyn, asg, grd)$ where:

– X is a finite set of real-valued variables.

– $\mathcal{M}$ is a finite set of discrete states, E is a transition relation $E \subseteq \mathcal{M} \times \Sigma \times \mathcal{M}$ and Σ a set of labels.

– $init$ and inv are functions which associate to each state a predicate over the variables in X. Letter $\mathcal{D}$ denotes the set $\{(M, v) \in \mathcal{M} \times \mathbb{R}^{|X|}) |\, v \vDash inv(M)\}$ where the expression $v \vDash inv(M)$ means that the predicate $inv(M)$ is satisfied in v.

– dyn is a function that associates to each state a predicate over the variables in X and the first derivatives of these variables. It is used to define the set of continuous evolutions which occur at each state (system of ODEs).

– asg is a function such that, given an edge $(e = (M_1, l, M_2) \in E)$, return a predicate over v_0, v_1, states of M_0 and M_1, respectively, after a discrete jump. This provides an assignment to each edge. Finally, the function grd associates each edge with a guard, i.e., a predicate over X.

We can see that this family of automata is a reconfigurable system where:

– The set X is the set of state variables.

– The set $\mathcal{V}$ is $\mathbb{R}$.

– Each state $M \in \mathcal{M}$ represents a configuration given by the solution of the system of ODEs obtained with $dyn(m)$ and its invariant D_M is the set $\{v|(M, v) \in \mathcal{D}\}$ and, therefore, $\mathbb{T} = \mathbb{R}_0^+$. In this way $M(v_0, t)$ is the state which assigns to each state variable the value of the respective solution for the Cauchy problem with the ODEs obtained from $dyn(M)$ and initial state v_0, after t units of time.

– The set $\mathcal{Q}$ of discrete events is given by the set containing all discrete events Q defined as $\{((M_0, v_0), (M_1, v_1)) |\, e = (M_0, l, M_1) \in E$, the predicate $asg(e)$ is valid over v_0, v_1 and the predicate $grd(e)$ is true over $v_0\}$ for each $l \in \Sigma$ and each pair $(M_0, M_1) \in \mathcal{M} \times \mathcal{M}$.

Finally, we note that the hybrid automaton consider $init$ as the possible initial states and values for the state variables. Although possible, we did not consider such set in our definition of reconfigurable systems since we will not use it.

The bouncing ball is a classical example of a hybrid system and we present it to illustrate how to consider it as a reconfigurable system.

Example 3.3 (Bouncing Ball)**.** A bouncing ball can be seen as a reconfigurable system and, as explained above, as a system with hybrid dynamics. Anytime the ball hits the ground and bounces back there is a change in the differential equations which drives the evolution of the position of the ball along time. Since this change occurs at discrete time instants we observe a reconfiguration at those instants. We recall the Figure 4 in order to describe the bouncing ball as a reconfigurable system $(X, \mathcal{V}, \mathcal{M}, \mathcal{Q})$ where:

– $X = \{v, h\}$

– $\mathcal{V} = \mathbb{R}$

– $\mathcal{M} = \{M\}$ is a singleton and $\mathbb{T} = \mathbb{R}_0^+$. For each t, $M((v_0, h_0), t)$ is the solution for the Cauchy problem: $v' = -g \wedge h' = v$, with initial values (v_0, h_0) at t. The invariant $D_M = \{(v, h) |\, h \geq 0\}$.

– $\mathcal{Q} = \{Q\}$ is a singleton such that $Q = \{((M, (\bar{v}, 0)), (M, (-c\bar{v}, 0))) |\, \bar{v} < 0\}$.

Now we focus on how to use d$\mathcal{L}$ proof calculus to study and prove properties of hybrid systems. In order to do this, we recall the hybrid programs obtained in Section 2 using the syntax of d$\mathcal{L}$. Considering

H the initial height and assuming the law of physics about kinetic and potential energy – $v^2 \leq 2g(H-h)$ – we can prove that the height h of the ball will always be between 0 and the initial height H. Fig. 5 shows the initial screen of KeYmaera for the proof of this property. Indeed, we only need to assume that the gravitational acceleration g is positive, the initial height H is positive, and that the value of c, an elasticity constant, is between 0 and 1.

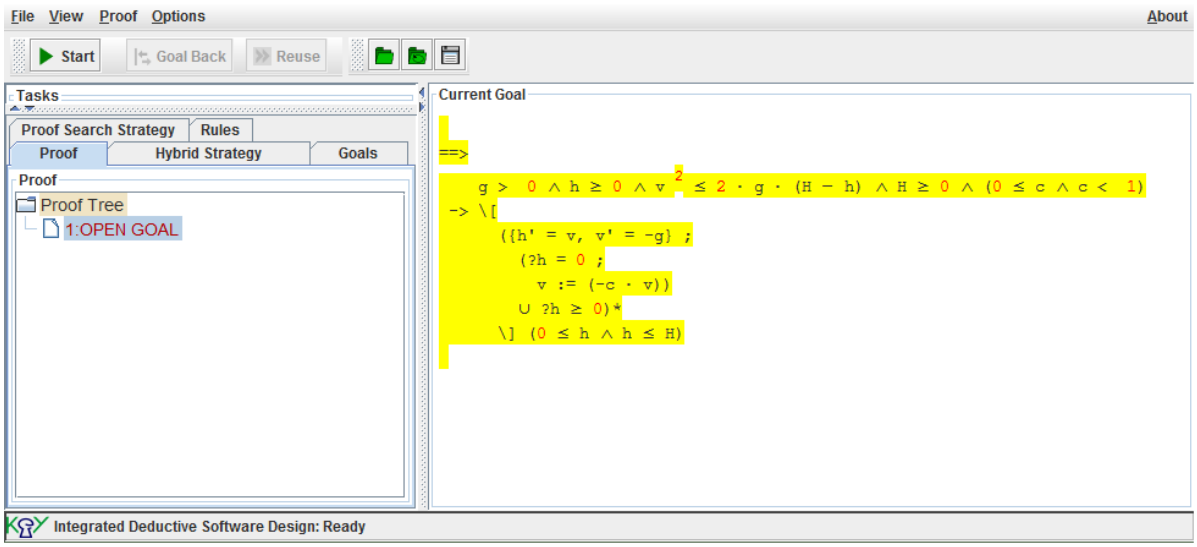

Figure 5: Initial screen of the bouncing ball problem in KeYmaera.

We note that, in this example, the exact value of g (gravitational acceleration) was not specified. In fact, d$\mathcal{L}$ proof calculus is able to prove properties and formulas containing variables (unlike, for example, model checking). In [11] there are many other examples essentially from mechanics. Next examples illustrate the reconfigurability in biological and chemical context.

Example 3.4 (Chemical Reconfigurable System)**.** There are many situations in real world that present reconfigurable behavior. In [6], Prassana De Silva discusses several examples in biological and chemical systems. He considers input/output systems and points out that modifying either the input or the output observed we can obtain the notion of reconfiguration in the behavior of the system. In particular, an example of a logic gate obtained from the response of a molecule to certain inputs is given in [6]. In that example, two different ions were introduced into an environment containing the referred molecule and the presence/absence of luminescence was observed. Depending on the ion introduced, different results were produced. Indeed, the logic gate obtained can vary between a YES logic gate and a PASS 0 logic gate (which are shown in Table 1) if the inputs used is, respectively, either Pb^{2+} or Cu^{2+}.

Input	YES	PASS 0
0	0	0
1	1	0

Table 1: Two different logic gates obtained in Example 3.1

Example 3.5 (A Boolean network with external input)**.** Here, we recall a Boolean network presented in [20] and further analyzed in [5]. It is a basic description of the circadian rhythm of cyanobacteria, in terms of two types of proteins (Kai A and Kai C): protein Kai A is represented by variable A, and protein Kai C may be present in four different forms, represented by variables S, T or TS (for protein

phosphorylated at different sites), and u (for unphosphorylated protein):

$$\begin{aligned} A^+ &= \neg S \\ T^+ &= A \wedge u \\ TS^+ &= A \wedge T \\ S^+ &= TS \end{aligned}$$

In this model, the evolution of the variables A,T,TS, and S follows a pre-defined set of rules, depending on each other and u, while the evolution of the unphosphorylated protein u is instead treated as an external input to the Boolean model. This u depends on the amount of protein Kai C translated at each instant; in this basic model, for simplicity, we study only the cases corresponding to high expression ($u \equiv 1$) or weak expression ($u \equiv 0$).

For each of these input values, the state transition graphs of the Boolean model represent M_1 and M_2 which are $\mathbb{Z}_0^+$-configurations and correspond to the two elements of set $\mathcal{M}$ in Definition 3.1. Also, X corresponds to the set of boolean variables $\{A, T, TS, S\}$, $\mathcal{V} = \{0, 1\}$ and $D_{M_1} = D_{M_2} = Sta(X)$. These are indeed different configurations since M_1, (corresponding to the case $u = 0$) contains a single state attractor (1000) and M_2 (corresponding to the case $u = 1$) contains a cyclic attractor with 7 states (synchronous case: $0000 \to 1000 \to 1100 \to 1110 \to 1111 \to 0111 \to 0001 \to 0000$).

Thus, if translation of Kai C is blocked, the system is at a stationary state and the circadian rhythm is disrupted; if protein Kai C is present, the system follows a sequence of phosphorylations which are at the basis of the circadian rhythm. Here, the set of discrete events contains two elements $\mathcal{Q} = \{Q_1, Q_2\}$ where $Q_1 = \{((M_2, x), (M_1, x))\}$ and $Q_2 = \{((M_1, x), (M_2, x))\}$ for any $x \in Sta(X)$. These two discrete events indeed contain all possible switches since there are only two possible input values for u: Q_1 occurs whether we change from $u = 1$ to $u = 0$ and Q_2 occurs whether we change from $u = 0$ to $u = 1$. This is easily realized by the "biologist" in the lab: for an initial state in some "configuration" M_i, either block or allow Kai C translation, Q_1 or Q_2, respectively.

These examples show how reconfigurability naturally comes across. Each configuration of the system depends on given combination of input variables. Moreover, this approach can also be used in systems whose dynamics (or set of worlds W) are described by differential equations. More specifically, reconfigurability can be understood as discrete modifications of the system of ODEs that guide the evolution of a system.

4. Applying d$\mathcal{L}$ in a biological hybrid context

A very popular framework used to model biological regulatory networks are those based on ODEs. In such models, variable x_i represents the concentration of species i, for $i = 1, \ldots, n$ (these may be proteins, enzymes, mRNAs, etc.). For each variable x_i, the corresponding differential equation describes the rate of increase/decrease in the concentration of species i. These changes in concentrations are induced by the network of interactions among all species (consisting typically of activations and inhibitions).

Activations and inhibitions are often represented by increasing or decreasing Hill functions:

$$\frac{x^m}{x^m + \theta^m} \quad \text{or} \quad \frac{\theta^m}{x^m + \theta^m}$$

where θ represents a threshold concentration and m represents a cooperativity effect. The sum of all direct interactions on species i can thus be written as a combination (sums of products) of Hill functions, into a function $f_i(x_1, \ldots, x_n)$. For simplicity, it is often assumed that each species has a linear natural degradation, $\gamma_i x_i$, which leads to equations of the form:

$$x_i' = f_i(x_1, \ldots, x_n) - \gamma_i x_i, \quad i = 1, \ldots, n. \tag{1}$$

Since these functions are not linear, it becomes hard to analytically study such models and piecewise linear (PWL) models are proposed as a simplification (see [21, 22] for more information). These more

abstract models may be obtained by letting the exponent m tend to infinity and replacing the Hill functions by step functions (see also [23]):

$$1 - s^-(x,\theta) = s^+(x,\theta) = \begin{cases} 1, & \text{if } x > \theta \\ 0, & \text{if } x < \theta. \end{cases}$$

These functions are not defined at the thresholds, so the equations $x_i' = f_i(x)$ must be defined as differential inclusions whenever $x_i = \theta_i$, for some i. In general, there may be several thresholds associated with each variable x_i $(\theta_i^1, \theta_i^2, \cdots)$. This class of systems is widely used to model biological regulatory networks. First introduced by Glass and co-authors [23] it was then extended (see, for instance, [21, 22, 24]) as a useful framework to obtain analytical results on differential equations. Following the above discussion, a piecewise linear system is defined on a family of regular domains $(\mathcal{X}_i)$ together with their boundaries $(\mathcal{D}_i)$, which form a partition of the state space:

$$\mathcal{X} = (\mathcal{X}_1 \cup \cdots \cup \mathcal{X}_\ell) \cup (\mathcal{D}_1 \cup \cdots \cup \mathcal{D}_\kappa), \quad \mathcal{X}_i \cap \mathcal{X}_j = \emptyset \quad \mathcal{D}_i \cap \mathcal{D}_j = \emptyset$$

with $\mathcal{X}_i$ and $\mathcal{D}_j$ disjoint two by two. The regular and boundary domains have the form:

$$\mathcal{X}_i = (\theta_1^{a_1}, \theta_1^{a_1+1}) \times \cdots \times (\theta_n^{a_n}, \theta_n^{a_n+1})$$
$$\mathcal{D}_i = \{x \in \mathcal{X} : \ x_k = \theta_k^a, \text{ some } k, \ x_j \in (\theta_j^b, \theta_j^{b+1}), \ j \neq k\}.$$

Any $\mathcal{D}_i$ is part of the boundary of one or more $\mathcal{X}_k$, so let $\mathcal{N}(\mathcal{D}_i)$ denote the set of all domains whose boundary contains $\mathcal{D}_i$ (its "neighbors"). For each regular domain the dynamics is determined by a continuous vector field in $\mathcal{F} = \{f_1, \ldots, f_\ell\}$ with $f_i : \mathcal{X} \to \mathcal{X}$. The piecewise linear system can now be written as an ODE in each regular domain

$$x' = f_j(x), \quad \text{if } x \in \mathcal{X}_j$$

and as a differential inclusion in each boundary domain (using $\bar{co}$ to denote the convex hull of two or more vector fields):

$$x' \in \bar{\text{co}}\{f_k(x) : \ \mathcal{X}_k \in \mathcal{N}(\mathcal{D}_i), \quad \text{if } x \in \mathcal{D}_i.$$

Note that, inside regular domains, the equations (1) become linear and can be explicitly solved. The global solution is still continuous and can be obtained by concatenating the solutions in each regular domain. Technical problems may arise at the boundary domains whenever two neighboring vector fields have opposite signs [22]. To avoid these technicalities, the examples given in this section have *transparent walls*, meaning that the solutions can be naturally continued at the boundaries, even though the vector fields have a discontinuity. The dynamical behavior of PWL can be studied in a general way, requiring only a set of inequalities between the parameters [25]. As illustrated next, PWL appear naturally as reconfigurable systems.

Example 4.1 (Piecewise linear systems in d$\mathcal{L}$)**.** This general class of systems can be interpreted as a class of reconfigurable systems with $X = \{x_1, ..., x_n\}$ being the set of state variables, $\mathcal{V} = \mathbb{R}$ and the set $\mathcal{M}$ containing ℓ configurations M_i, $i \in \{1, ..., \ell\}$ given by the dynamics in each regular domain. $M_i : \mathcal{X}_i \times \mathbb{R} \to Sta(X)$ is such that $M_i(x_0, 0) = x_0$ and $\frac{dM_i(x,t)}{dt} = f_i(x)$. The set of discrete events $\mathcal{Q}$ contains an element Q_i for each boundary $\mathcal{D}_i$, $k = 1, \ldots, \kappa\}$. To determine the possible transitions, it is necessary to evaluate the convex hull of the neighboring vector fields: $Q_k = \{\big((M_i, x), (M_j, x)\big) : \mathcal{X}_i, \mathcal{X}_j \in \mathcal{N}(\mathcal{D}_k), x \in \mathcal{D}_k$ and $\bar{\text{co}}\{f_i, f_j\}$ points from $\mathcal{X}_i$ to $\mathcal{X}_j\}$.

As mentioned above, there may be some special cases (such as sliding modes or fixed points along the boundary domains) where a transition is not so straightforward to compute. However, this doesn't affect reconfigurability: depending on the case, a new configuration corresponding to the sliding mode or fixed point might be added to $\mathcal{M}$.

Therefore, we can use the syntax of d$\mathcal{L}$ to specify the dynamics of PWL systems. To do this we can apply the two kinds of atomic programs considered by d$\mathcal{L}$: the differential equations specify the dynamics within each domain and the discrete jump sets along with the remaining operators of dynamic logic are used to specify the changes between domains described by the discrete events.

A given system may exhibit several reconfigurability "layers", that is, several functions q_i representing different actions on the systems. This is next illustrated by adding a control input to PWL systems.

Example 4.2 (Piecewise linear systems with discrete inputs in d$\mathcal{L}$)**.** The setup is the same as in Example 4.1 but now each vector field f_i depends on external parameters, $\mu = (\mu_1, \ldots, \mu_r)$, taking values in a discrete set: $\mu \in U = \{u_1, \ldots, u_p\}^r$ and $f_i : \mathcal{X} \times U \to \mathcal{X}$. In regular domains, the piecewise linear system becomes

$$x' = f_j(x; \mu), \quad \text{if } x \in \mathcal{X}_j, \mu \in U$$

and in each boundary domain x' is defined as a differential inclusion. There are r input parameters, each taking p possible values so one can define a set of $\ell \times p^r$ configurations M_{ij} associated to the vector fields $f_i(\cdot; \mu^j)$ with $1 \leq i \leq \ell$ and $1 \leq j \leq p^r$. The set of discrete events contains an element for each boundary domain and one for each change in the input value, of one of the forms: $Q = ((M_{ij}, x), (M_{i\tilde{j}}, x))$ for all $j \neq \tilde{j} \in \{1, \ldots, p^r\}$ or $Q_k = \{\big((M_{ij}, x), (M_{\tilde{i}j}, x)\big) : \mathcal{X}_i, \mathcal{X}_{\tilde{i}} \in \mathcal{N}(\mathcal{D}_k),\ x \in \mathcal{D}_k \text{ and } \bar{\text{co}}\{f_i(\cdot; \mu^j), f_{\tilde{i}}(\cdot; \mu^j)\}$ points from $\mathcal{X}_i$ to $\mathcal{X}_{\tilde{i}}\}$. A PWL system with inputs is thus a reconfigurable system and can also be described by the syntax of d$\mathcal{L}$. Then the set of general transitions will be given by all possible pairs $(\mathcal{X}_i, \mu)$ where the value of μ changes for any $i = 1, ..., \ell$ and $(\mathcal{D}_i, \mu_j)$, for fixed μ_j and $i = 1, \ldots, \kappa$. A PWL system with inputs is a reconfigurable system and can also be described by the syntax of d$\mathcal{L}$.

Example 4.3 (A PWL negative loop with inputs)**.** Let us consider a theoretical regulatory network composed of two species, forming a negative loop with auto-regulation and an external discrete controller μ to regulate the expression of y:

$$\begin{cases} x' = 5\dfrac{y^m}{2^m + y^m} + \dfrac{x^m}{3^m + x^m} - x \\[2ex] y' = 5\dfrac{3^m}{3^m + x^m} + \dfrac{y^m}{y^m + 2^m} + \mu - y. \end{cases}$$

To simplify, we consider, in the sequel, that the degradation rate of the substance represented by μ is extremely high and is reduced to zero quickly if the control fades.

The corresponding piecewise linear model is the one presented in Table 2, according to its four regular domains labeled B_i, $i = 1, \ldots, 4$ with $B_1 = \{0 < x < 3, 0 < y < 2\}$, $B_2 = \{0 < x < 3, 2 < y < 6\}$, $B_3 = \{3 < x < 6, 2 < y < 6\}$, and $B_4 = \{3 < x < 6, 0 < y < 2\}$.

$\begin{cases} x' = 5 - x \\ y' = 6 + \mu - y \end{cases}$ $x < 3 \wedge y > 2$	$\begin{cases} x' = 6 - x \\ y' = 1 + \mu - y \end{cases}$ $x > 3 \wedge y > 2$
$\begin{cases} x' = -x \\ y' = 5 + \mu - y \end{cases}$ $x < 3 \wedge y < 2$	$\begin{cases} x' = 1 - x \\ y' = \mu - y \end{cases}$ $x > 3 \wedge y < 2$

Table 2: Piecewise linear model with a discrete control variable μ.

As in Example 4.2, this model presents two forms of reconfigurable behavior. These reconfigurations occur whenever the system transits between regular domains or whenever the discrete input μ changes its value. The input parameter is a scalar ($r = 1$) and its discrete values for instance $\mu \in \{0, 1, 2, \ldots, 10\}$ ($p = 11$).

A careful analysis of the model in Table 2 shows that, in the case $\mu \equiv 0$, the system asymptotically converges to an orbit centered at $(x, y) = (3, 2)$ as shown in Figure 6.

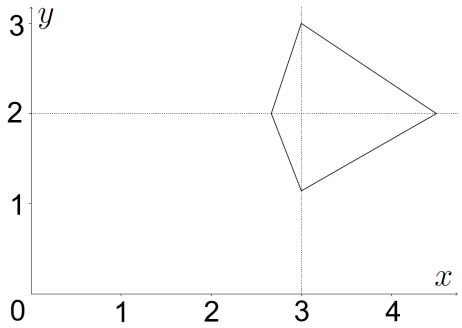

Figure 6: Orbit of the system.

However, suppose the objective is to drive the system toward a state where both proteins are highly expressed, that is, the trajectory remains in the domain B_3. This can be done if we set the control as $\mu = 2$ if $x \geq 3$ and $y \geq 2$, and $\mu = 0$ otherwise. Indeed, we can use the d$\mathcal{L}$ logic to prove that the resulting system has a stable steady state at $(x, y) = (6, 3)$. In order to be able to do this, we need to describe this property by a formula of d$\mathcal{L}$. Firstly, we must write the hybrid program that describes the evolution of this biological system in the semantics of d$\mathcal{L}$. Since there are four domains, we describe the evolution on each. Note that, in order to simplify the calculations, the boundaries are included in specific domains, without changing the example. We also introduce a time counter τ:

$$ctrl_1 \equiv (?x \leq 3 \wedge y \leq 2; \mu := 0; (x' = -x, y' = 5 - y + \mu, \tau' = 1 \,\&\, x \leq 3 \wedge y \leq 2))$$
$$ctrl_2 \equiv (?x \leq 3 \wedge y \geq 2; \mu := 0; (x' = 5 - x, y' = 6 - y + \mu, \tau' = 1 \,\&\, x \leq 3 \wedge y \geq 2))$$
$$ctrl_3 \equiv (?x \geq 3 \wedge y \leq 2; \mu := 0; (x' = 1 - x, y' = -y + \mu, \tau' = 1 \,\&\, x \geq 3 \wedge y \leq 2))$$
$$ctrl_4 \equiv (?x \geq 3 \wedge y \geq 2; \mu := 2; (x' = 6 - x, y' = 1 - y + \mu, \tau' = 1 \,\&\, x \geq 3 \wedge y \geq 2))$$

For instance, the execution of the program $ctrl_1$ first verifies if the actual state is in the respective domain defined by $x \leq 3 \wedge y \leq 2$; if so, then it executes $\mu := 0$ and proceeds with the continuous evolution defined by the respective system of differential equations. In the system of differential equations, we add a differential equation referring to the time counter τ. Also, this evolution can only be executed while the system still are in the same domain (this constraint is represented by "& $x \leq 3 \wedge y \leq 2$"). The executions of $ctrl_2$, $ctrl_3$ and $ctrl_4$ are analogous.

Then, the evolution of the complete system is described by:

$$bioctrl \equiv (ctrl_1 \cup ctrl_2 \cup ctrl_3 \cup ctrl_4)^*$$

The execution of *bioctrl* can determine which subprogram must be executed. In order to consider behaviors that come from one domain to another, the Kleene operator is added. Also, to allow this, in the hybrid program presented before, the boundaries are considered on each domain.

If a state is a steady state, then there exists a neighborhood such that any other state inside that neighborhood converges asymptotically to the steady state. In particular, we can conclude that some state is a steady state if there is a disk centered in that state such that if we choose another state x inside it, the distance of this state to the center will decrease along the trajectory obtained by the evolution of the system. We can describe this property by the following formula of d$\mathcal{L}$:

$$\exists c > 0(\forall\, 0 < k < c((x-6)^2 + (y-3)^2 = k \wedge\ \tau = 0 \rightarrow [bioctrl](\tau = 0 \vee (x-6)^2 + (y-3)^2 < k)))$$

where c and k are logical variables and τ, x, y and u are state variables.

We point out that the time counter τ is important here because, by definition, the hybrid program *bioctrl* can terminate without executing the continuous evolution. In this case, the state would not vary and the distance to the stable steady state $(x, y) = (2, 3)$ would not decrease. Therefore, we state that, after the execution of *bioctrl*, either no continuous evolution occurred (and, thus, $\tau = 0$) or the distance to the state $(x, y) = (2, 3)$ decreased.

At this point, we must note that, in general, there are asymptotic stable steady states which do not verify the property above. Indeed, in a general case, the distance can increase during some periods of time along the trajectory to an attractor. However, since we are considering a piecewise linear models, we know these phenomena do not occur.

We can construct a proof within the d$\mathcal{L}$ proof calculus to show that the property above holds. However, this is an extensive proof and we prefer not present it in this paper; it can be found in [26]. This proof could be partially performed using the referred software *KeYmaera*. Although any proof can be done "by hand", we can use this semi-automathic software to save much time.

Example 4.4 (Cyanobacteria circadian rhythm)**.** Recall the example introduced in Section 3 on the cyanobacteria circadian rhythm, presented in [20]. In that paper, an ODE model was also studied and parameters estimated. Gathering all data obtains the following ODE model for the circadian rhythm of cyanobacteria:

$$\begin{cases} x_a' = 10\,\dfrac{5^4}{x_s^4+5^4} - 0.45x_a \\ x_t' = 20.51\,\dfrac{(205.43-x_t-x_{ts}-x_s)^4}{(205.43-x_t-x_{ts}-x_s)^4+29.95^4}\cdot\dfrac{x_a^4}{x_a^4+10^4} - 0.24x_t \\ x_{ts}' = 10.74\,\dfrac{x_t^4}{x_t^4+11.42^4}\cdot\dfrac{x_a^4}{x_a^4+10^4} - 0.28x_{ts} \\ x_s' = 6.61\,\dfrac{x_{ts}^4}{x_{ts}^4+10.16^4}\cdot\dfrac{13^4}{x_a^4+13^4} - 0.081x_s \end{cases}$$

As it is, this model is not a reconfigurable system, since it can be described by a single world. Thus, to obtain hybrid dynamics, a control variable u can be added. Let u be a control which induces the production of the protein whose concentration is represented by x_a. In order to include this variable we change the differential equation associated to x_a to $x_a' = 10\,\dfrac{5^4}{x_s^4+5^4} + u - 0.45x_a$. Moreover, we consider an additional differential equation with the following control strategy:

$$u' = \begin{cases} 1-u, & \text{if } x_a \leq 15 \\ -u, & \text{otherwise.} \end{cases}$$

Let $X = (x_a, x_t, s_{ts}, x_s, u)$ and $F : \mathbb{R}^5 \to \mathbb{R}^5$ such that $F(X) = (F_1, F_2, F_3,\ F_4, F_5)$, where F_i corresponds to the i^{th} equation of the ODE model with the control u, $i \in \{1,2,3,4,5\}$. We can represent our model by the expression $X' = F(X)$. Additionally, if we consider $F_5 = a - u$, where a can either be 0 or 1, according to the control strategy, we can describe the dynamics of this model by a hybrid program of d$\mathcal{L}$. To be realistic, a discrete controller would check the value of x_a in regular intervals of time in order to decide which action has to be taken. Let us consider that the discrete controller checks the state of the model every 0.1 units of time. We can describe the dynamics of this model by the following hybrid program, which we denote by *hybdpgrm*:

$$subpgrm \equiv \Big(\tau := 0; \big((?x_a < 15); a := 1 \cup (?x_a \geq 15); a := 0\big); \big(\tau' = 1, X' = F(X)\ \&\ \tau \leq 0.1\big)\Big)$$

$$hybdpgrm \equiv subpgrm; \big((?\tau = 0.1); subpgrm\big)^*$$

The hybrid program denoted by *subpgrm* starts by considering a time counter state variable (τ) which will be needed further. Then it verifies if either $x_a < 15$ or $x_a \geq 15$ and adjusts the value of a, accordingly. Thereafter, it runs the continuous evolution constrained by $\tau \leq 0.1$ in order to guarantee that the continuous evolution will not run longer than what is allowed between two successive verifications by the discrete controller.

Hence, the program *hybdpgrm* executes the program *subpgrm* successively. It begins by executing it once. Afterwards, it verifies if the discrete controller must check the value of a (this is known by performing the test $?\tau = 0.1$). If so, it runs *subpgrm* again. We introduce the operator * to obtain a hybrid program which successfully terminates on all reachable states of the original model.

Now that we have a hybrid program describing the evolution of the hybrid model (ODE with discrete controller), we can test formulas expressing properties we want to prove. For instance, we can ask if, for the initial point $X = (17, 10, 10, 10, 0)$, it is true that x_a is always greater than 13. This property is expressed by the formula:

$$?(x_a = 17 \wedge x_t = 10 \wedge x_{ts} = 10 \wedge x_s = 10 \wedge u = 0) \rightarrow [hybdpgrm]x_a > 13$$

In this Section, we have shown that it is possible to apply d$\mathcal{L}$ in a biological context to formally prove features of such systems (as the existence of steady states) and how to use computational tools to reason about hybrid systems. Indeed, the syntax of d$\mathcal{L}$ is very expressive and properties which are usually studied in biological systems, such as the existence of attractors or the existence of positively invariant regions, can be expressed in d$\mathcal{L}$ language. In this way, d$\mathcal{L}$ is a potentially useful tool in the study of biological systems.

5. Conclusion and future work

This work aims at applications in the interdisciplinary branch of biology and engineering of synthetic biology. The concept of reconfigurability (well known in Computer Science) is discussed in the context of models for physical and biological systems. Some general properties are identified as naturally leading to reconfigurable systems: first, the existence of a discrete controller or external input to change the system by instantaneous jumps and, second, the existence of different dynamics (i.e., velocities or vector fields) in different regions of the state space. Both cases belong to a family of hybrid systems, those described by continuous dynamics but subject to discrete events that induce discontinuous switches in the vector fields. In particular, the class of piecewise linear systems, widely used in biological models, falls into this family.

Reconfigurable systems can be studied with a variety of logical tools and formalisms. Here, we emphasize the use of differential dynamic logic as a tool to reason about biological regulatory networks; d$\mathcal{L}$ allows us to represent continuous models without specifying values for the free parameters which is not possible in model checking. This allows to test properties involving those parameters.

This paper enhances the usefulness of d$\mathcal{L}$ to approach biological systems with discrete controllers as hybrid systems. The examples presented here illustrate some elementary problems and pave the way for analysis of more complex models and, furthermore, the software *KeYmaera* can reduce, in a significant way, the amount of effort required to complete the proofs. The syntax and semantics of this logic is able to specify several control problems, for example (i) specify that some set is positively invariant; (ii) specify that some control procedure guarantee that the system will reach a desired configuration; and (iii) specify that some control procedure will avoid an undesirable configuration, for instance. Furthermore, the procedure of d$\mathcal{L}$ proof calculus often cindicates how to find counterexamples or additional premises needed to prove a desirable formula. In practice, this may lead to the calculation of, for instance, basins of attraction or control strategies. These are very promising perspectives for the analysis of piecewise linear systems: in fact, many regulatory questions can be posed in terms of finding a trajectory that will lead from one specific domain to another [24, 27]. A similar case is presented in Example 4.3, where the objective was to highly express both proteins.

On the computational side, we would like to point out that *KeYmaera* calls WOLFRAM MATHEMATICA in order to solve ODEs. Because of this, although *KeYmaera* is able to solve many problems relating to models nowadays used in the study of biological systems, its computational power is limited to the one of WOLFRAM MATHEMATICA. Hence, we would like to endow *KeYmaera* with dedicated ODEs solvers (for example, optimized for Piecewise Linear ODEs) and, by this way, increase the efficiency of the tool. For instance, we consider that would be interesting to obtain a symbiosis between

KeYmaera and BIOCHAM or GNA (see [25]). We think that this can be done in the same lines as it was done in [28] for interfacing *Averest* and *KeYmaera*.

Another limitation on the use of *KeYmaera* concerns the proof of formulas with the existential modal operator. There are other software tools, see for example [29], that seem more appropriated for this kind of properties. We are currently working on testing an alternative to *KeYmaera*. In future, we intend to obtain a symbiosis between these two approaches.

Acknowledgments. This work was partially supported by the ERDF European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia within project POCI-01-0145-FEDER- 016692 and project UID/MAT/04106/2013 at CIDMA. DF acknowledges the support by FCT via the PhD scholarship PD/BD/114186/2016. MC was supported in part by the French agency for research through project ANR-16-CE33-0016-01.

References

[1] A. Marr, Growth rate of *Escherichia coli*, Microbiol. Rev. 55 (1991) 316–333.

[2] E. Ozbudak, M. Thattai, H. Lim, B. Shraiman, A. van Oudenaarden, Multistability in the lactose utilization network of *Escherichia coli*, Nature 427 (2004) 737–740.

[3] A. Madeira, Foundations and techniques for software recongurability, Ph.D. thesis, Universities of Aveiro, Minho and Porto (MAP-i adjoint doctoral program) (2013).

[4] A. Madeira, R. Neves, L. S. Barbosa, M. A. Martins, A method for rigorous design of reconfigurable systems, Science of Computer Programming 132, Part 1 (2016) 50 – 76.

[5] D. Figueiredo, Relating bisimulations with attractors in boolean network models, in: International Conference on Algorithms for Computational Biology, Springer, 2016, pp. 17–25.

[6] A. P. De Silva, Molecular logic-based computation, no. 12, Royal Society of Chemistry, 2012.

[7] A. Goñi-Moreno, M. Amos, A reconfigurable nand/nor genetic logic gate, BMC systems biology 6 (1) (2012) 126.

[8] A. Tamsir, J. J. Tabor, C. A. Voigt, Robust multicellular computing using genetically encoded nor gates and chemical/wires/', Nature 469 (7329) (2011) 212–215.

[9] C. G. Cassandras, S. Lafortune, Introduction to Discrete Event Systems, 2^{nd} Ed., Springer, New York, 2008.

[10] M. Bernardo, P. Degano, G. Zavattaro (Eds.), Formal Methods for Computational Systems Biology, Vol. 5016 of Lecture Notes in Computer Science, Springer, 2008.

[11] A. Platzer, Logical analysis of hybrid systems: proving theorems for complex dynamics, Springer Science & Business Media, 2010.

[12] Y. Kouskoulas, D. Renshaw, A. Platzer, P. Kazanzides, Certifying the safe design of a virtual fixture control algorithm for a surgical robot, in: Proceedings of the 16th international conference on Hybrid systems: computation and control, ACM, 2013, pp. 263–272.

[13] A. Platzer, E. M. Clarke, Formal verification of curved flight collision avoidance maneuvers: A case study, Springer, 2009.

[14] A. Platzer, J.-D. Quesel, European train control system: A case study in formal verification, in: Formal Methods and Software Engineering, Springer, 2009, pp. 246–265.

[15] G. Batt, C. Belta, R. Weiss, Temporal logic analysis of gene networks under parameter uncertainty, IEEE Transactions on Automatic Control 53 (Special Issue) (2008) 215–229.

[16] P. Blackburn, M. De Rijke, Y. Venema, Modal Logic: Graph. Darst, Vol. 53, Cambridge University Press, 2002.

[17] D. Harel, D. Kozen, J. Tiuryn, Dynamic logic, MIT press, 2000.

[18] D. Harel, J. Tiuryn, D. Kozen, Dynamic Logic, MIT Press, Cambridge, MA, USA, 2000.

[19] T. A. Henzinger, The theory of hybrid automata, in: Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996, 1996, pp. 278–292.

[20] M. Chaves, M. Preto, Hierarchy of models: From qualitative to quantitative analysis of circadian rhythms in cyanobacteria, Chaos: An Interdisciplinary Journal of Nonlinear Science 23 (2) (2013) 025113.

[21] H. De Jong, Modeling and simulation of genetic regulatory systems: a literature review, Journal of computational biology 9 (1) (2002) 67–103.

[22] R. Casey, H. de Jong, J. Gouzé, Piecewise-linear models of genetic regulatory networks: equilibria and their stability, J. Math. Biol. 52 (2006) 27–56.

[23] L. Glass, S. Kauffman, The logical analysis of continuous, nonlinear biochemical control networks, J. Theor. Biol. 39 (1973) 103–129.

[24] L. Habets, P. J. Collins, J. H. van Schuppen, Reachability and control synthesis for piecewise-affine hybrid systems on simplices, IEEE Transactions on Automatic Control 51 (6) (2006) 938–948.

[25] H. De Jong, J. Geiselmann, C. Hernandez, M. Page, Genetic network analyzer: qualitative simulation of genetic regulatory networks, Bioinformatics 19 (3) (2003) 336–344.

[26] D. Figueiredo, Differential dynamic logic and applications, Master's thesis, University of Aveiro (2015).

[27] M. Chaves, J. Gouzé, Exact control of genetic networks in a qualitative framework: the bistable switch example, Automatica 47 (2011) 1105–1112.

[28] X. Li, K. Bauer, K. Schneider, Interactive verification of cyber-physical systems: Interfacing averest and keymaera, in: Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on, IEEE, 2013, pp. 1405–1412.

[29] S. Kong, S. Gao, W. Chen, E. Clarke, dReach: δ-Reachability Analysis for Hybrid Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2015, pp. 200–205.